AD-A081 308   MARYLAND UNIV  COLLEGE PARK COMPUTER SCIENCE CENTER   F/G 12/1
PARALLEL COMPUTATION OF CONTOUR PROPERTIES.(U)
DEC 79  T DUBITZKI, A ROSENFELD, A Y WU   SAFOSR-77-3271
UNCLASSIFIED  TR-848   AFOSR-TR-80-0143   NL

1 OF 1
AD
A081308

END
DATE
FILMED
4-80
DTIC

# COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

TR-848
AFOSR-77-3271

December, 1979

# PARALLEL COMPUTATION OF CONTOUR PROPERTIES

Tsvi Dubitzki
Angela Y. Wu
Azriel Rosenfeld

Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, MD 20742

## ABSTRACT

Some contour properties can be derived in parallel by a string or cycle of automata in linear time, faster than can be done with a single processor. In particular the intersection points of two contours, the straightness of a line, the union or intersection of two contours, and polygonal approximations of a contour are computed in linear time.

The support of the U.S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Eleanor Waters and Kathryn Riley in preparing this paper.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) Unclassified

|  |  |
|---|---|
| REPORT DOCUMENTATION PAGE | READ INSTRUCTIONS BEFORE COMPLETING FORM |
| 1. REPORT NUMBER AFOSR TR- 80-0143 / 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle) PARALLEL COMPUTATION OF CONTOUR PROPERTIES. | 5. TYPE OF REPORT & PERIOD COVERED Interim rept. |
|  | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s) Tsvi/Dubitzki, Azriel/Rosenfeld, Angela Y./Wu | 8. CONTRACT OR GRANT NUMBER(s) AFOSR-77-3271 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS University of Maryland, Computer Science Center, College Park, MD 20742 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM, Bolling AFB, Washington, D. C. 20332 | 12. REPORT DATE Dec[redacted]979 |
|  | 13. NUMBER OF PAGES 26 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) | 15. SECURITY CLASS. (of this report) UNCLASSIFIED |
|  | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing
Pattern recognition
Cellular automata

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Some contour properties can be derived in parallel by a string or cycle of automata in linear time, faster than can be done with a single processor.

In particular the intersection points of two contours, the straightness of a line, the union or intersection of two contours, and polygonal approximations of a contour are computed in linear time.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

412318

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

## 1. Introduction

A cycle automaton is a cyclically connected string of N processors, i.e., the last processor is connected to the first one. One of the processors is designated as the distinguished processor D.

The power of this type of cellular automaton lies in its parallel computation capability and in the option of rotating information cyclically, thus shortening the time of information propagation and comparison.

In this paper we study the computation of properties of a contour by a cycle automaton that initially stores the chain code of the contour. Parallelism is used to speed up the computation of straightness, curvature, polygonal approximations, convexity, intersection points, and the union or intersection of two contours.

Note that if the cycle automaton stores an open curve then the chain code leading back from the Nth processor to the distinguished processor D is null.

For some of the computation algorithms, it is necessary for each processor to know its coordinates relative to D. The coordinates are easily found from the chain codes but each processor must have memory at least O(log N) in order to compute and store them.

A

## 2. Extrema and inflection points

Finding inflection or extremum points on a curve can be done sequentially in linear time since they are local properties. Using parallelism, extrema cannot be found in faster than linear time. Nevertheless, we define these processes for a cycle automaton storing the chain code of a curve so that we can use these methods for further computations in the following sections.

Let a curve of N points be stored in a cycle automaton L of N processors. Let C represent the 8-adjacency (3-bit) chain code of a point A of L, $C_{p(A)}$ its predecessor's chain code and $C_{s(A)}$ its successor's chain code. The distinguished node D starts a signal along L. Upon arriving at a processor, say A, the signal causes it to check if A is an inflection point or extremum.

The criterion for an inflection point at A is (see Fig. 1(a)):

1) $C_A \neq C_{p(A)}$

2) There is a point B such that $C_i = C_{s(i)}$ for $A \leq i < p(B)$ but $C_B \neq C_{p(B)}$ and $C_B \neq (C_{p(B)}+4)$ modulo 8.

3) $|C_B - C_{p(A)}| \leq 2$ or $(8-|C_B - C_{p(A)}|) \leq 2$

If the arc AB contains more than 2 points then the inflection point is at the midpoint of the arc AB. Processor A finds that midpoint as follows: A sends a unit speed signal and a 1/3 speed signal towards B. The unit speed signal bounces back from B and meets the 1/3 speed signal at the middle of AB. The original signal is prevented from looking for inflection points as long as conditions 2 and 3 are checked.

A simple criterion for an extremum at A is (Fig. 1(b)): $C_{p(A)} \neq C_A$, i.e. any change in the direction of the curve is an extremum. If we are dealing with a simple contour where the interior of the contour is always on one side when we traverse it in a given direction, then it is possible to determine which extremum point is a minimum and which is a maximum with respect to the interiors. For example: if we traverse the contour with the interior always on our right side then the condition for a maximum is (for 8-adjacency) $C_i = C_{p(i)}$+5,6, or 7 (modulo 8), and the condition for a minimum is $C_i = C_{p(i)}$+ 1,2, or 3 (modulo 8).

## 3. Straight line detection

Let L be a digital arc. A necessary condition for L to be a digitization of a straight line is to be composed of alternating runs of points (a run is a collection of consecutive points in one direction) where the runs have only two different directions that differ by 45°, and for one of these directions, the run length must be 1. This property can be detected by a cycle automaton storing the chain code of the curve in linear time. Upon verifying that the curve satisfies this property we rotate it to lie in the first octant of the plane ($0 \le \alpha \le 45°$) and to have the distinguished node as the leftmost node. Let a leftmost or a rightmost point of a run of length >1 in the first octant be called a left or right step-node respectively (see Fig. 2).

It is proved in [5] that L represents a straight line iff it satisfies the chord property, i.e. any point on a chord joining two points on L is at less than unit distance from L. Formally, if $(x,y)$ are the coordinates of a point on the chord, then there exists a point $(i,j)$ on L such that $\max(|x-i|,|y-j|)<1$. Clearly one only needs to check the chords whose endpoints are both left ends, or both right ends, of runs; and one only needs to check those intermediate points that lie above right run ends, or below left run ends. Now consider a point A which is the left end of a run, and the chord AB which has maximal slope among all the chords AQ such that Q is a left step node and lies to the right of A. If AB is near L, then every right step node between points A and B is within distance 1 from any of these AQ's. For any point P to the right of B, the chord BP lies above AP since the slope of AP is less than the slope of AB. Thus chord BP near L implies that AP is near L between B and P. The chords joining right step nodes have analogous properties.

By using a cycle automaton of N processors, we can determine whether the curve is a straight line by checking the chord property at all the run end nodes along the curve. This is done as follows: D, the distinguished node, which is the leftmost node of the cycle automaton, sends a signal that causes the processors to compute their coordinates (assuming D has coordinates (0,0)) and also marks the left and right step nodes in L (see Fig. 2). As soon as a left end node knows its coordinates, it propagates them to the left. Whenever any left step node A receives the coordinates of another left step node B it computes the pair $(x_A-x_B, y_A-y_B)$ which represents the slope of the line connecting A and B. Each time A receives the coordinates of another left step node it updates its pair so as to keep the pair which gives the maximum slope. When A receives the signal from the rightmost node, it has the slope of the steepest chord at A. A starts to propagate the slope pairs rightward together with a counter (initialized to zero) which increments by 1 for each move from node to node. Upon receiving such a pair a node computes the distance between itself and the line represented by the slope, i.e. if the inspected node has the coordinates (x,y) and the incoming pair is $(\Delta x_i, \Delta y_i)$, the node checks whether $\left|\frac{\Delta y_i}{\Delta x_i} \cdot \text{counter}_i - y\right| < 1$. If this condition is violated a rejection signal is sent to D meaning that the digitized curve L is not a straight line. The propagation of a pair $(\Delta x_i, \Delta y_i)$ stops at the node where $\text{counter}_i = \Delta x_i$, that is, when the pair arrives at the farthest node giving the slope represented by $(\Delta x_i, \Delta y_i)$.

When D propagates its maximal slope pairs to the right, it also sends a unit speed (if we assume distance conditions can be checked in one time step) timing signal to the right, which bounces back at the rightmost node. If the timing signal returns to D without any rejection signal, D starts a similar process of computing minimal slopes between rightmost points of the runs (right step nodes) and looking for violation of the chord property below L. If this time too no rejection signal comes back to D, L is declared to be a straight line. Clearly the above procedure is a linear time algorithm. Detecting straightness using one processor takes $O(N^2)$ time steps.

## 4. Intersection

### 4.1 Self intersection and touching of a curve

Given a cycle automaton storing the chain code of a curve, we can determine whether the curve intersects itself and where. The distinguished processor D which has the coordinates (0,0) sends a signal around the curve which causes each node i to compute its coordinates $(x_i, y_i)$ from the coordinates of its predecessor node i-1. Upon coming back to D the signal causes the chain code and the coordinates of the nodes together with the chain code of their predecessors to go around the cycle. Each processor i compares the incoming coordinates of processor j to its own to decide whether it is a self-intersection or a self-touching node. There are two cases:

1) coordinates (i) = coordinates (j)

Let $h = \min(C_i, \overline{C}_{p(i)})$ and $k = \max(C_i, \overline{C}_{p(i)})$ where $\overline{C} = C+4 \pmod 8$ for 8-adjacency. If $C_j$ and $\overline{C}_{p(j)}$ are both inside the closed interval $[h,k]$ or both outside the open interval $(h,k)$, then the curve touches itself at the point i (see Fig. 3). In the other cases we have a self-intersection at the point i.

2) coordinates (i) ≠ coordinates (j)

We have a self-intersection at A and B (see Fig. 4) when

$$x_A = x_B,\ x_{p(A)} = x_{p(B)}$$

$$y_A = y_{p(B)},\ y_B = y_{p(A)}$$

or analogously with x and y interchanged.

The propagation of the coordinates and the comparisons take O(N) time steps after which each intersection or touching node is marked.

Looking for self-intersections along a curve without using a cycle automaton would take $O(N^2)$ time steps as the computations performed by the N processors in parallel have to be done in sequence by one processor: N stages, each of O(N) time steps.

## 4.2 Intersection of two curves

Let the first curve (L1) have distinguished node D1 and N1 processors. Similarly the second curve (L2) has D2 and N2. D1 and D2 are connected and know their own coordinates in a common coordinate system. D1 and D2 start sequential processes along their own cycle automata to compute the coordinates of all the nodes (processors) along their cycles using their coordinates and the chain codes. When the coordinate computation signal comes back to D1 it starts pushing the coordinates, the chain code and the predecessor's chain code of its own cycle nodes into a special channel of L2 through D2. The same is done by L2 to L1. At this phase each processor compares its own coordinates with those coming in the special channel. Using the same criteria as in Section 4.1, one can mark the intersections and touching nodes.

This process takes a total of O(N1+N2) time steps after which D1 or D2 can output the coordinates of their intersection points in linear time. Sequentially the process would have taken O(N1·N2) time steps. Storing and passing the coordinates of the processors in the cycle automata needs augmented memory of O(log N1+log N2).

## 4.3 Inclusion or exclusion of two simple contours

In this section we study when a contour is completely surrounded by, i.e. in the interior of, another contour. Unfortunately it is not always clear what is the interior of a contour if it has a self-intersection. A simple contour is a closed curve without self-intersection. Consider the contours in Fig. 5. The 8-shaped contour of Fig. 5(a) has a well-defined interior region. It can be turned into a simple curve by turning the intersection point into a touching point without changing the region enclosed by the contour. But in Fig. 5(b) we don't know whether the internal loop belongs to the region or is outside it. If we turn the intersection point here into a touching point we will make the internal loop part of the outside of the region. As we are unable to detect locally which kind of intersection point we have we will deal here only with contours without self-intersections, i.e. simple contours.

Inclusion of two simple contours is determined by checking their intersection. If they do not intersect at all then either one contour includes the other or they exclude each other. This last fact can be determined in principle by choosing a point, say D2 in L2, and drawing a one way line from D2 to infinity. Then we count the number of intersection points of that line with L1. If that number is odd then L1 surrounds L2, otherwise either L2 surrounds L1 or L2 excludes L1. Thus if the count is even we also have to draw a one way line from D1 to infinity and count the number of intersection points of this line with L2. In case this number is even then L1 excludes L2, otherwise L2 includes L1. (Zero is considered even.)

Specifically, D2 sends its own y coordinate to D1. D1 sends this $y_{D2}$ around L1 with a counter initialized to zero. In sequence the signal compares $y_{D2}$ to the coordinates of all the points of L1. Upon finding a point $P \in L1$ which satisfies $y_P = y_{D2}$ the counter is incremented by 1 only in case there is an intersection point at P, i.e. the y coordinate of the successor of P is different from the y coordinate of either P or its predecessor (otherwise we have a touching point). When it comes back to D1 the signal has the number of intersection points of the line (starting at D2) with L1. In case the count is even the same is done by D2. Finding the intersection points between contours takes O(N1+N2) time steps. Counting the number of intersection points of the line starting at D2 takes O(N1) time steps, and counting those of the line starting at D1 takes O(N2) time steps, thus a total of O(N1+N2) time steps.

Finding the chain code of that part of L2 which is inside L1 is almost equivalent to finding the intersection of L1 and L2 and is discussed in Section 5.3.

## 5. The union and intersection chain codes of two contours

In this section we show how to determine the chain code of the border of the union or intersection of two simply-connected regions.

Let L1 and L2 be two cycle automata storing the chain codes of two simple contours. Let D1 and D2 be the distinguished nodes of L1 and L2. D1 and D2 are joined. Note that if we traverse curve L1 then consecutive intersection points of L1 with another curve L2 appear sequentially along the curves L1 and L2.

Before computing the union or intersection chain code we should reconfigure the two cycle automata (L1 and L2) so that the common intersection points and touching points at both these curves will be joined. This can be done in O(N1+N2) where N1 and N2 are the number of processors in L1 and L2 respectively.

### 5.1 The reconfiguration process

Let A be the first intersection point on L1 if we follow the chain code starting at D1 (Fig. 6(a)). Then A connects itself directly with D1 and D1 connects itself to the corresponding intersection point (A in Fig. 6(a)) in L2 via D2. Finally A in L1 connects itself directly with A in L2. All the temporary connections except the connection D1 $\rightarrow$ D2 are erased during the reconnection process. Then B in L1 connects itself to B in L2 using the existing connection A(L1) $\rightarrow$ A(L2) and so do all the other intersection and touching nodes.

### 5.2 The chain code of the union

Assume that both in L1 and L2 the chain code is clockwise, i.e. if we traverse L1 or L2 along the chain code the interior of the closed region is on our right side. If this is not the case then D1 and D2 can communicate and make the necessary modifications to the chain code in one or both of L1 and L2. Also assume that each node knows which of its neighbors is its successor along either contour and that at intersection points of L1 with L2, L1 knows which direction of L2 leads to its interior and vice-versa.

First we check if L1 and L2 intersect at all by the methods of Section 4.1. If not we find out by the method of Section 4.3 whether one contour includes the other and output the chain code of that appropriate contour as the chain code of L1 $\cup$ L2. If L1 and L2 do intersect then D1 and D2 send signals along L1 and L2 respectively which find the leftmost node on each contour. Next D1 exchanges information with D2 about the node chosen by each of them and the node that is the leftmost of the two is chosen and marked by either D1 or D2 (whichever found it). That marked node, say X on L1 (Fig. 6(a)), is on the outer border of L1 $\cup$ L2 and therefore we start the traversal of L1 $\cup$ L2 here. The traversing signal in state p goes along L1 following the chain code until it reaches the first intersection or touching node with L2 (e.g. A in Fig. 6(a)) and marks it with w. Then it switches to state q and turns to follow L2 in a direction that leaves the interior of L2 on the same side that it was while following L1. It is clear that we must switch to follow L2 as we are looking for the chain code of L1 $\cup$ L2. The signal follows L2 until it reaches another intersection or touching point with L1, marks it with w, switches back to state p and continues along the chain code of L1. This process repeats until the signal comes back to X, the leftmost point of L1. While traversing L1 $\cup$ L2 the signal pushes back towards D1 the chain code it encounters along L1 and L2. The chain code is output in sequence through D1.

The traversal of the above signal takes O(N1+N2) time steps. It remains to output the chain codes of the holes that might be generated by the intersection of L1 and L2 (see Fig. 6(a)). This is done as follows: X starts a new signal in state p traversing L1 in the same direction as before. The signal looks for an unmarked intersection node or a singly marked (with w) touching node. If the signal finds such a node it marks the node with v, switches to state q and follows L1 further until it reaches the next intersection or touching node. It marks the node with w, switches to L2 and follows L2 in a direction that leaves the outside of L2 on the same side that it was before with respect to the signal traversing L1. This guarantees that we are going to surround a region outside the regions enclosed by L1 and L2, i.e. a hole. Upon coming back to L1 at a node marked v the signal changes the mark v to w, switches back to state p and follows L1 again in the previous direction in search for an unmarked intersection node or a singly marked (with w) touching node. While in state q the signal pushes back toward D1 the chain code of the curve segment that it traverses. Note that finally each intersection node is marked once with w and each touching node twice with w.

At most (N1+N2) time steps after the signal comes back to D1 all the chain codes of the holes have been output through D1.

## 5.3 <u>The chain code of the intersection</u>

As defined in the previous section we assume that the chain code is clockwise and that each node of L1 or L2 knows which neighbor is its successor in the direction of the chain code. Also assume that all intersection or touching nodes are found. Again we use L1 and L2 reconfigured as in Section 5.1. If there are no intersection points of L1 with L2 we determine whether one contour contains the other (see Section 4.3), and output the chain code of the inside contour. If L1 and L2 exclude each other the intersection is null. Otherwise let X be the leftmost node of L1 and L2 and assume that X is on L1 (Fig. 6(a)) as in Section 5.2.

A clockwise signal p starts at X on L1 and follows its chain code. Whenever the signal meets an intersection of L1 with L2 it follows the branch of the intersecting contour that enters the region enclosed by the currently followed contour. The same is done with respect to touching nodes, that is, if a curve touches the currently followed contour from the side of its enclosed region then the traversing signal switches to that curve that bounces back to the interior of the above enclosed region. But if the touching node is on the outside of the region enclosed by the currently followed contour then the signal ignores it.

Each time the signal turns to follow L2 it marks the connection from L1 into L2 so as to avoid a repeated switch to L2 along that path. Once the signal p following L1 changes to signal q to follow L2, it starts pushing back towards D1 the chain code it traverses. The signal switches back to p when it wants to turn into L2 at an intersection or a touching point but finds that turn marked, i.e. the signal has just completed traversing the border of a connected component of the intersection.

This process repeats itself until the signal, in state p, visits X again and stops. Within O(N1+N2) time units after that the last chain code of the intersection contour is output through D1.

The process is illustrated by Fig. 6(b) where two intersection contours: CABC and DCD are output.

Note that finding the intersection or union of L1 and L2 would have taken O(N1·N2) time if done sequentially.

## 6. Curvature

[1] describes two basic methods of computing the curvature of a curve. These methods can be made faster by storing the chain code of the curve in a cycle automaton.

The k curvature method [1] involves computing the angle defined at a point on a curve, between the average direction, $V_i^{(1)}$, of the curve segment preceding that point and the curve segment succeeding that point, $V_i^{(2)}$. In other words:

$$V_i^{(1)} = \sum_{j=1}^{k} W_j \; V_{i-j}$$

$$V_i^{(2)} = \sum_{j=1}^{k} W_j \; V_{i+j-1}$$

where $V_i$ is the chain code of the curve at point i and $W_j$ are weights. The angle between $V_i^{(1)}$ and $V_i^{(2)}$ is:

$$\theta_i = \text{arc cos}\left(\frac{V_i^{(1)} \cdot V_i^{(2)}}{|V_i^{(1)}| \; |V_i^{(2)}|}\right)$$

The arc/chord distance method [1] draws the chord connecting each point i on a curve with the point i+k along that curve and finds the maximum distance between the curve i,i+k and the chord $\overline{i,i+k}$. This distance is a measure of the curvature at the point i. The above maximum distance can be computed by the method discussed in [2] and takes O(k) time steps. Using cycle automata these methods can be carried out partially in parallel. For instance for the arc/chord distance method a signal propagates sequentially starting from the distinguished node of the cycle automaton and forcing each point to compute the maximum distance between the arc i,i+k and the chord $\overline{i,i+k}$. If the cycle automaton has N processors (sampling points) then this process takes $O(N+C(k))$ time steps where $C(k)$ is a function of k. Doing the above computation sequentially with one processor would take $O(C(k) \cdot N)$ time steps. The same argument applies to the k-curvature method. The degree of the nodes in the cycle automaton is bounded by 4: one connection to a predecessor, one to a successor, one for the outgoing chord and one for the incoming chord. Note that if all the processors of the cycle automata were synchronized beforehand (all in the same initial state), then the curvature computation would have taken constant time.

## 7 . Polygonal approximation

Approximating a polygon means representing it by a subset of its vertices. Polygonal approximation is defined with respect to some criterion of optimality. We shall define it to be that approximating polygon (of a given number of nodes) which is closest in its area to the area bounded by the true polygon.

Let AB (Figure 7) be the polygonal approximation of the segment AEDCB. Then a one step optimal refinement of $\overline{AB}$ is ADB sine $h_D > h_E$ and $h_D > h_C$, i.e., the area of the triangle ADB is bigger than the area of any other triangle based on AB and having its third vertex on the segment AEDCB. Unfortunately this idea of choosing as a refinement the vertex farthest from the chord approximating a segment is proved optimal only for convex segments. Therefore we will deal either with convex polygons or with general polygons where we are given the concavity points or look for them by finding minima (Section 1). We will start our polygonal approximation with the set of concavity points so that each segment considered for refinement is convex. The way to look for a vertex farthest from a chord connecting the endpoints of a polygonal segment is described in [2] and takes time linear in the number of vertices in the segment.

Using a cycle automaton L which stores the chain code of a polygon makes it possible to perform the polygonal approximation refinement in parallel at all the refined segments. Suppose we have a convex polygon to approximate. At first D, the distinguished node of L, sends a unit speed signal and a 1/3 speed signal along L. The unit speed signal bounces back upon arriving cyclically at D and meets the 1/3 speed signal at the processor which is in the middle of the cycle automaton (A in Fig. 8). In parallel that middle processor sends clockwise and counterclockwise signals along L. Each one of these signals is looking for that point along the polygon which is farthest from the chord DA or AD. These points (B and C in Fig. 6) also send clockwise and counterclockwise signals looking for refinement vertices.* Thus if L contains N vertices the above parallel process will take $\frac{3}{2}N$ steps for finding the middle point and at most $O(\frac{N}{2}) + O(\frac{N}{4}) + \ldots O(\frac{N}{2^{\log N}})$ steps for the refinement. It sums to:

$$\frac{3}{2}N + N(1-\frac{1}{2^{\log N}}) = \frac{3}{2}N + N-1 = \frac{5}{2}N - 1$$

Refining the polygonal approximation sequentially might take $N^2$ time steps in the worst case.

If we have a concave polygon to approximate then we would start the polygonal approximation with the set of vertices located at the concavities, synchronize them (e.g., by means of the firing squad method [6]) and start the refinement process in parallel in the convex segments.

*This process continues to whatever degree of refinement we need and all the vertices chosen up to that point constitute the current polygonal approximation.

## 8. The area of a polygon

The computation of the area enclosed by a contour is done by a cycle automaton in time linear in the number of points in the contour, just as it is done by a single processor. Then the area enclosed by a contour of $N$ points is given by the trapezoidal rule

$$S = \sum_{i=1}^{N} a_{ix}(y_{i-1}+\frac{1}{2}a_{iy})$$

where $y_i = y_{i-1} + a_{iy}$

In fact each processor computes its own contribution to this sum, adds it to the partial sum and propagates it further along the line. Upon coming back to D the computing signal has sum S. This process takes augmented memory at the processors.

## References

[1] W. S. Rutkowski, Azriel Rosenfeld, A comparison of corner detection techniques for chain coded curves. TR-623, Computer Science Center, University of Maryland, January 1978.

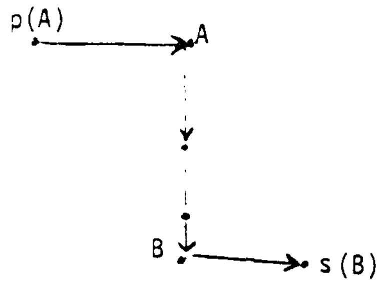
[2] H. Freeman, Computer processing of line drawing images, Computing Surveys 6, 1974, 57-97.

[3] Larry S. Davis, Shape representation and matching, Ph.D. thesis, University of Maryland, 1977.

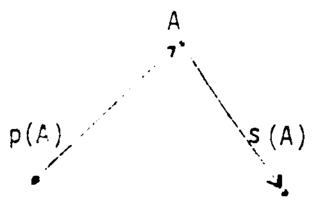
[4] Azriel Rosenfeld, A note on cycle grammars. TR-300, Computer Science Center, University of Maryland, April 1974.

[5] Azriel Rosenfeld, Digital straight line segments, IEEE Transactions on Computers 23, 1974, 1264-1269.

[6] F. R. Moore, E. G. Langdon, A generalized firing squad problem. Information and Control 12, 1968, 212-220.



(a) Inflection point



(b) Extremum

Figure 1.

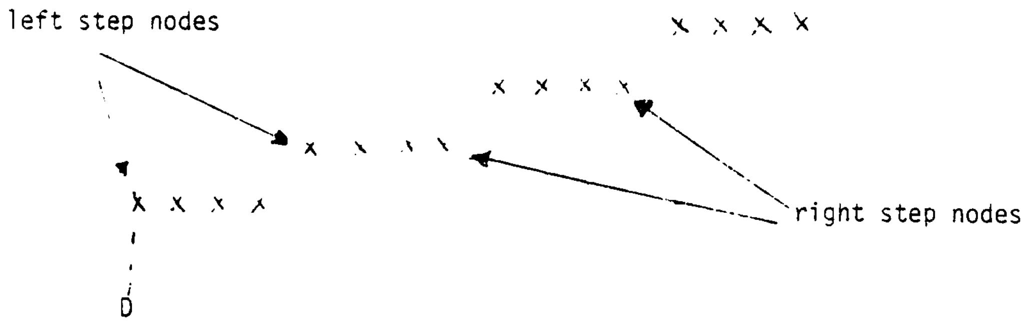


Figure 2: A digitized straight line.

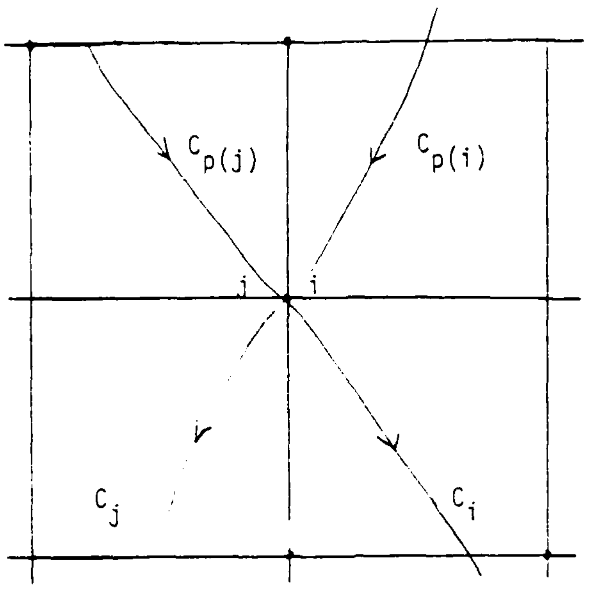


Figure 3:

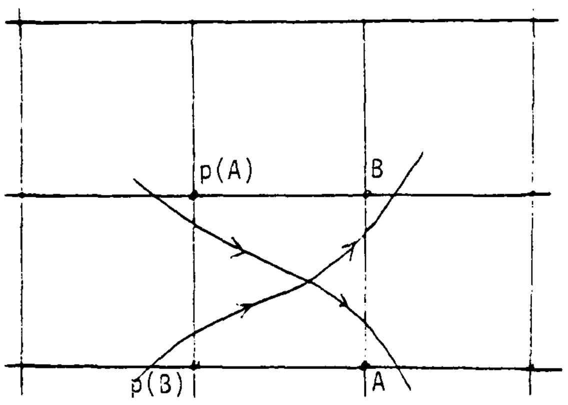


Figure 4:

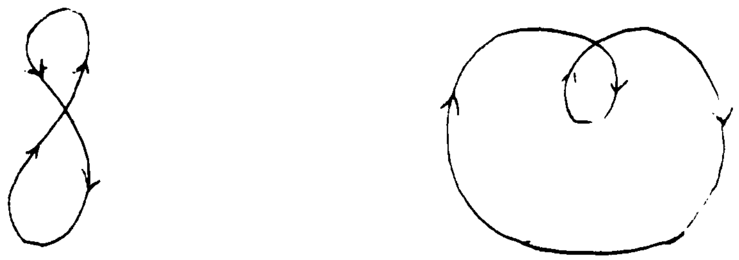
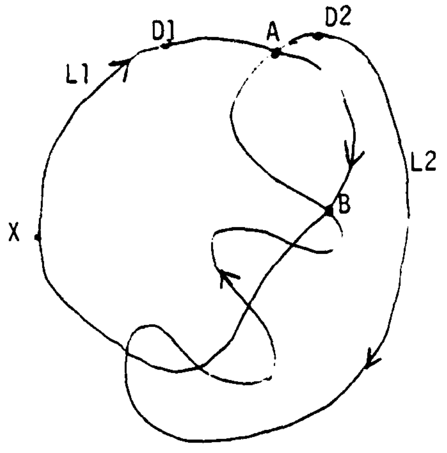
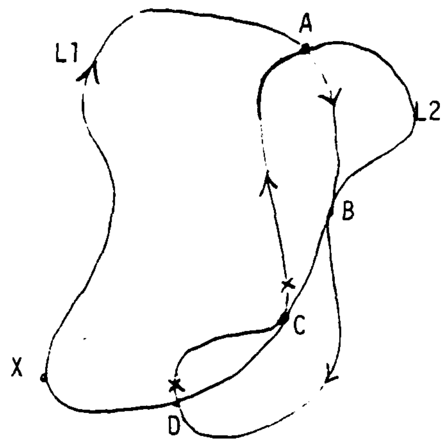


Figure 5



(a): intersection points



(b): touching and intersection points

Figure 6

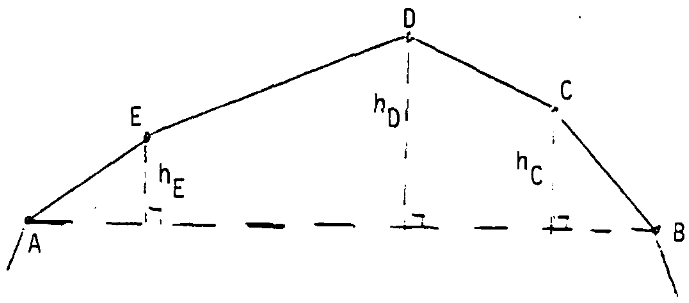


Figure 7.

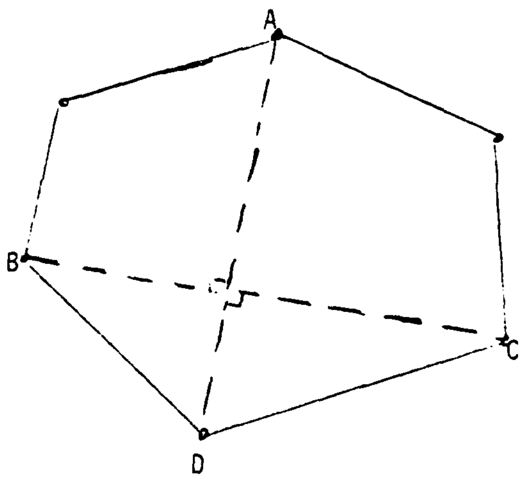


Figure 8.